



АКАДЕМИЯ СТРОИТЕЛЬСТВА И АРХИТЕКТУРЫ  
ДОНСКОГО ГОСУДАРСТВЕННОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ  
КВАЛИФИКАЦИИ

Кафедра «Информационные системы в строительстве»

## **МЕТОДИЧЕСКИЕ УКАЗАНИЯ**

к проведению практических занятий  
по дисциплине

# **«Моделирование объектов в OpenGL»**

Автор

Кокарева Я.А.

Ростов-на-Дону, 2016

## Аннотация

Методические указания предназначены для студентов, обучающихся по направлению 09.03.01 «Информатика и вычислительная техника», профиль подготовки – «Системы автоматизированного проектирования»

## Авторы



к.т.н., доцент кафедры  
«Информационные системы в строительстве»  
Кокарева Я.А.



## Оглавление

<b>ВВЕДЕНИЕ.....</b>	<b>4</b>
<b>Синтаксис команд.....</b>	<b>4</b>
<b>Практическая работа №1. Инициализация OpenGL .....</b>	<b>5</b>
<b>Практическая работа №2. Примитивы OpenGL .....</b>	<b>11</b>
<b>Практическая работа №3. Изменение предопределенных свойств примитивов .....</b>	<b>14</b>
<b>Практическая работа №4. Процедура для рисования окружности и правильных многоугольников .....</b>	<b>18</b>
<b>Практическая работа №5. Графики функций.....</b>	<b>21</b>
<b>Практическая работа №6. Аффинные преобразования ...</b>	<b>22</b>
<b>Практическая работа №7. Отображение трехмерных фигур.....</b>	<b>24</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....</b>	<b>36</b>

## ВВЕДЕНИЕ

Моделирование геометрических объектов является неотъемлемой частью создания систем автоматизированного проектирования. В качестве стандарта 3D графики признана кроссплатформенная открытая графическая библиотека OpenGL.

OpenGL - Open Graphics Library, открытая графическая библиотека. Термин "открытый" означает независимый от производителей. Библиотека завоевала огромную популярность и интегрирована со множеством языков и систем разработки приложений.

Библиотека OpenGL представляет из себя интерфейс программирования трехмерной графики. Единицей информации является вершина, из них состоят более сложные объекты. Программист создает вершины, указывает как их соединять (линиями или многоугольниками), устанавливает координаты и параметры камеры и ламп, а библиотека OpenGL берет на себя работу создания изображения на экране. OpenGL идеально подходит для программистов, которым необходимо создать небольшую трехмерную сцену и не задумываться о деталях реализации алгоритмов трехмерной графики. Для профессионалов, занимающихся программированием трехмерной графики, библиотека тоже будет полезной, т.к. она представляет основные механизмы и выполняет определенную автоматизацию.

OpenGL непосредственно не поддерживает работу с устройствами ввода, такими как мышь или клавиатура, т.к. эта библиотека является платформенно независимой. Но можно задействовать функции конкретной операционной системы, под которую пишется программа, или воспользоваться надстройками над OpenGL, такими как библиотеки GLUT или GLAUX.

Практические занятия по моделированию проводятся с использованием программной среды Delphi, которая не требует подключения дополнительных библиотек, кроме стандартной (поставляемой с драйверами видеокарты), для вывода на экран результатов программирования.

## СИНТАКСИС КОМАНД

Для того чтобы команды OpenGL были доступны в проекте, необходимо указать библиотеку в списке используемых модулей.

Все команды начинаются с префикса **gl**, затем идёт имя ко-



## Моделирование объектов в OpenGL

манды, цифра и суффикс. Цифра в окончании соответствует количеству аргументов, буква показывает требуемый тип аргумента.

Если имя команды заканчивается на *v* (векторная форма), то аргументом её служит указатель на массив значений. Например, если последние три символа в имени команды *3fv*, то её аргумент – адрес массива трёх вещественных чисел.

В общем виде команду можно представить:

*glCommandName {1,2,3,4} {b, s, i, f, d, ub, us, ui} {v} (arguments)*

Таблица 1. Возможные типы аргументов команд OpenGL

Символ	Обозначение типа в OpenGL	Расшифровка
b	GLbyte	Байтовый
s	GLshort	Короткий целый
i	GLint	Целый
d	GLdouble	Вещественный двойной точности
f	GLfloat	Вещественный
ub	GLubyte	Байтовый, беззнаковый
us	GLushort	Короткий целый, беззнаковый
ui	GLuint	Целый, беззнаковый

Почти всегда предпочтительно использовать команду в вещественной форме, поскольку хранит данные OpenGL именно в вещественном формате.

## ПРАКТИЧЕСКАЯ РАБОТА №1. ИНИЦИАЛИЗАЦИЯ OPENGL

Для инициализации библиотеки OpenGL в Delphi необходимо добавить ссылку на библиотеку в используемые модули, а также прописать некоторые специфические процедуры для корректного определения формата пикселей и отображения на форме изображения.

Ниже представлены два варианта инициализации. В первом используются события (Events): OnCreate, OnDestroy, OnPaint.

## Моделирование объектов в OpenGL

Изображение программируется в глобальной процедуре `FormPaint`, там же задаются основные параметры видовых и проективных матриц, а также освещения и материалов. Во втором используются события `OnCreate`, `OnResize`, `OnClose`. Параметры матриц, освещения, материалов определены в процедурах `FormCreate`, `FormResize`, а сами команды рисования прописываются в локальной процедуре `Draw`. Процедура `FormResize` отвечает за перерисовку изображения при изменении размеров окна.

### 1-й вариант

```
unit Unit1;
```

### **interface**

#### **uses**

```
Windows, Messages, SysUtils, Variants, Classes, Graphics,  
Controls, Forms, Dialogs, OpenGL;
```

#### **type**

```
TForm1 = class(TForm)  
  procedure FormPaint(Sender: TObject);  
  procedure FormCreate(Sender: TObject);  
  procedure FormDestroy(Sender: TObject);  
private  
  { Private declarations }  
public  
  { Public declarations }
```

#### **var**

```
Form1: TForm1;  
hrc: HGLRC;
```

#### **implementation**

```
{ $R *.dfm }
```

```
// процедура установки формата пикселей
```

```
procedure SetDCPixelFormat (hdc : HDC);
```

#### **var**

```
pfd      :      TPixelFormatDescriptor;
```

## Моделирование объектов в OpenGL

```
nPixelFormat : Integer;  
begin  
  FillChar (pfd, SizeOf (pfd), 0);  
  nPixelFormat := ChoosePixelFormat (hdc, @pfd);  
  SetPixelFormat (hdc, nPixelFormat, @pfd);  
end;  
  
procedure TForm1.FormCreate(Sender: TObject);  
begin  
  SetDCPixelFormat(Canvas.Handle);  
  hrc := wglCreateContext(Canvas.Handle);  
  WindowState := wsMaximized; // установлен размер окна во  
  весь экран  
end;  
  
procedure TForm1.FormPaint(Sender: TObject);  
begin  
  wglMakeCurrent(Canvas.Handle, hrc);  
  glClearColor (0.5, 0.5, 1, 1.0); // установлен цвет фона в  
  формате RGBA  
  glMatrixMode(GL_PROJECTION); // установлена проектив-  
  ная матрица  
  glLoadIdentity(); // установлена единичная матрица  
  gluOrtho2D(-10,10,-5,5); // установка двумерной коорди-  
  натной сетки внутри окна:  $x_{\min}=-10$ ,  $x_{\max}=10$ ,  $y_{\min}=-5$ ,  $y_{\max}=5$   
  glClear (GL_COLOR_BUFFER_BIT); // очистка буфера  
  
  //здесь будут команды рисования  
  
  wglMakeCurrent(0, 0);  
end;  
  
procedure TForm1.FormDestroy(Sender: TObject);  
begin  
  wglDeleteContext(hrc);  
end;  
  
end.
```

2-й вариант



```
unit Unit1;
```

## interface

### uses

```
Windows, Messages, Forms, Classes, Controls, ExtCtrls,  
ComCtrls, StdCtrls, Dialogs, SysUtils, OpenGL;
```

### type

```
TForm1 = class(TForm)  
  procedure FormCreate(Sender: TObject);  
  procedure FormResize(Sender: TObject);  
  procedure FormClose(Sender: TObject; var Action: TClose-  
Action);
```

### private

```
  ghRC: HGLRC;  
  ghDC: HDC;  
  procedure Draw;  
end;
```

### var

```
  Form1: TForm1;
```

## implementation

```
{$R *.DFM}
```

```
{+++ Формат пикселя +++}
```

```
function bSetupPixelFormat(DC: HDC): boolean;
```

### var

```
  pfd: PIXELFORMATDESCRIPTOR;  
  ppfd: PPIXELFORMATDESCRIPTOR;  
  pixelformat: integer;
```

### begin

```
  ppfd := @pfd;
```

```
  ppfd.nSize := sizeof(PIXELFORMATDESCRIPTOR);
```

```
  ppfd.nVersion := 1;
```

```
  ppfd.dwFlags := PFD_DRAW_TO_WINDOW or  
PFD_SUPPORT_OPENGL or PFD_DOUBLEBUFFER;  
  8
```





## Моделирование объектов в OpenGL

```
ppfd.dwLayerMask := PFD_MAIN_PLANE;
ppfd.iPixelFormat := PFD_TYPE_RGBA;
ppfd.cColorBits := 32;
ppfd.cDepthBits := 8;

ppfd.cAccumBits := 0;
ppfd.cStencilBits := 0;

pixelformat := ChoosePixelFormat(dc, ppfd);
if pixelformat = 0 then
begin
  MessageBox(0, 'ChoosePixelFormat failed', 'Error', MB_OK);
  bSetupPixelFormat := FALSE;
  exit;
end;

if not SetPixelFormat(dc, pixelformat, ppfd) then
begin
  MessageBox(0, 'SetPixelFormat failed', 'Error', MB_OK);
  bSetupPixelFormat := FALSE;
  exit;
end;

bSetupPixelFormat := TRUE;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
```

```
  ghDC := GetDC(Handle);
  if not bSetupPixelFormat(ghDC) then Close();
  ghRC := wglCreateContext(ghDC);
  wglMakeCurrent(ghDC, ghRC);
  glClearColor(0.5, 0.5, 1, 1.0); // установка цвета фона в
```

```
формате RGBa
```

```
  FormResize(Sender);
```

```
end;
```



## Моделирование объектов в OpenGL

```
procedure TForm1.FormResize(Sender: TObject);  
begin  
  glViewport(0, 0, Width, Height);  
  glMatrixMode(GL_PROJECTION);  
  glLoadIdentity();  
  gluOrtho3D(-1,1,-1,1,-1,1); // координатная сетка для трех-  
мерного пространства (Xmin, Xmax, Ymin, Ymax, Zmin, Zmax)  
  gluLookAt(0.2,0.2,-0.1,0,0,0,0,0,1); //задание положения и  
ориентации камеры (будет рассмотрена ниже)  
  glMatrixMode(GL_MODELVIEW);  
end;  
  
procedure TForm1.FormClose(Sender: TObject; var Action:  
TCloseAction);  
begin  
  if ghRC <> 0 then  
    begin  
      wglMakeCurrent(ghDC,0);  
      wglDeleteContext(ghRC);  
    end;  
  if ghDC <> 0 then ReleaseDC(Handle, ghDC);  
end;  
  
procedure TForm1.Draw;  
begin  
  glClear(GL_COLOR_BUFFER_BIT);  
  
  // команды рисования  
  
  SwapBuffers(ghDC);  
  
end;  
  
end.
```

## ПРАКТИЧЕСКАЯ РАБОТА №2. ПРИМИТИВЫ OPENGL

Примитивы создаются следующим образом:

***glBegin(GLenum mode);*** // открываем операторские скобки. Указываем в качестве аргумента примитив

***glVertex[2 3 4][s i f d](...);*** // указываем первую вершину примитива

***...*** // остальные вершины

***glVertex[2 3 4][s i f d](...);*** // последняя вершина примитива

***glEnd();*** // закрываем операторские скобки

**Примечание.** Между командами ***glBegin(GLenum mode)*** и ***glEnd()*** можно располагать несколько однотипных примитивов, таких как: отдельные точки, отдельные отрезки, треугольники, четырехугольники.

Обычно, вершины задаются одним из четырех способов.

***glVertex2d(x,y);*** // две переменных типа double

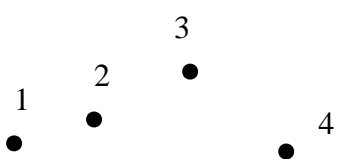
***glVertex3d(x,y,z);*** // три переменных типа double

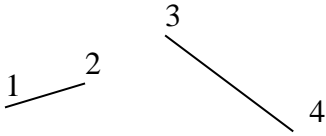
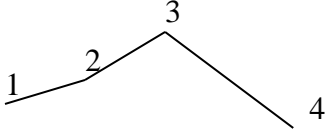
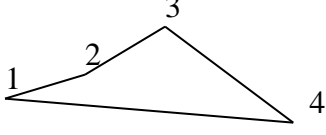
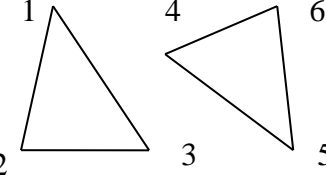
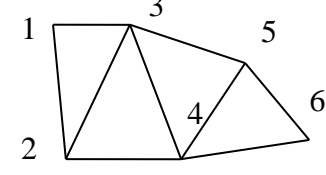
***glVertex2dv(array);*** // массив из двух переменных типа double

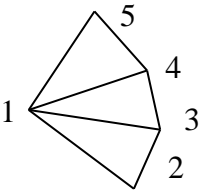
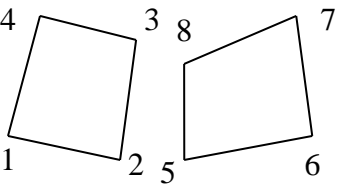
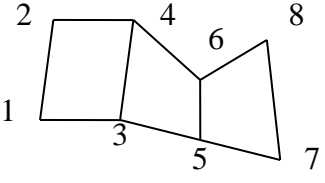
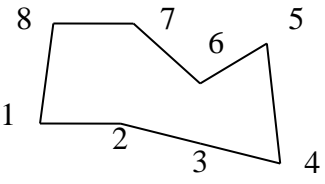
***glVertex3dv(array);*** // массив из трех переменных типа double

Параметр *mode* может принимать одно из значений, приведенных в табл. 1.

Таблица 1. Значения параметра *mode*

Значение	Смысл
GL_POINTS  	Обрабатывает каждую вершину как отдельную точку. Вершина <i>n</i> определяет точку <i>n</i> .

<p>GL_LINES</p> 	<p>Обработывает каждую пару вершин как независимый линейный сегмент. Вершины <math>2n-1</math> и <math>2n</math> определяют прямую <math>n</math>. Рисуется <math>N/2</math> прямых.</p>
<p>GL_LINE_STRIP</p> 	<p>Рисует связанную группу линейных сегментов от первой вершины до последней. Вершины <math>n</math> и <math>n+1</math> определяют прямую <math>n</math>. Рисуется <math>N-1</math> прямых.</p>
<p>GL_LINE_LOOP</p> 	<p>Рисует связанную группу линейных сегментов от первой вершины до последней, а затем назад к первой. Вершины <math>n</math> и <math>n+1</math> определяют прямую <math>n</math>. Последняя прямая определена вершинами 1 и <math>N</math>. Рисуется <math>N</math> прямых.</p>
<p>GL_TRIANGLES</p> 	<p>Обработывает тройку вершин как независимый треугольник. Вершины <math>3n-2</math>, <math>3n-1</math> и <math>3n</math> определяют треугольник <math>n</math>. Рисуется <math>N/3</math> треугольников.</p>
<p>GL_TRIANGLE_STRIP</p> 	<p>Рисует связанную группу треугольников. Для нечетного значения <math>n</math> вершины <math>n</math>, <math>n+1</math> и <math>n+2</math> определяют треугольник <math>n</math>. Для четного значения <math>n</math> вершины <math>n+1</math>, <math>n</math> и <math>n+2</math> определяют треугольник <math>n</math>. Рисуется <math>N-2</math> треугольников.</p>

<p>GL_TRIANGLE_FAN</p> 	<p>Рисует связанную группу треугольников. Вершины 1, <math>n+1</math> и <math>n+2</math> определяют треугольник <math>n</math>. Рисуются <math>N-2</math> треугольников.</p>
<p>GL_QUADS</p> 	<p>Обрабатывает каждую группу из четырех вершин в качестве независимого четырехугольника. Вершины <math>4n-3</math>, <math>4n-2</math>, <math>4n-1</math> и <math>4n</math> определяют четырехугольник <math>n</math>. Рисуются <math>N/4</math> четырехугольников.</p>
<p>GL_QUAD_STRIP</p> 	<p>Рисует связанную группу четырехугольников. Вершины <math>2n-1</math>, <math>2n</math>, <math>2n+2</math> и <math>2n+1</math> определяют четырехугольник <math>n</math>. Рисуются <math>N/2-1</math> четырехугольников.</p>
<p>GL_POLYGON</p> 	<p>Рисует отдельный выпуклый многоугольник. Вершины от 1 до <math>N</math> определяют этот многоугольник.</p>

Примеры задания точек:

```
glVertex2s(2, 3) – точка с координатами  $x = 2, y = 3, z = 0,$ 
 $w = 1;$ 
glVertex3d(0.0, 0.0, 3.14) – точка с координатами  $x = 0.0, y$ 
 $= 0.0, z = 3.14, w = 1.0;$ 
GLdouble dvect[3] = {5.0, 9.0, 1992.0};
glVertex3dv(dvect) – точка с координатами  $x = 5.0, y = 9.0, z$ 
 $= 1992.0, w = 1.0.$ 
```

**Пример.** Изобразить треугольник с вершинами  $(-2,3)$ ,  $(4,4)$ ,  $(7,-4)$ .

Для изображения треугольника воспользуемся значением параметра `GL_TRIANGLES`. Программирование всех изображений будет происходить в процедуре `FormPaint` (первый вариант инициализации) или `Draw` (второй вариант) после команды очистки буфера `glClear (GL_COLOR_BUFFER_BIT)`.

```
procedure TForm1.FormPaint(Sender: TObject);  
begin  
    wglMakeCurrent(Canvas.Handle, hrc);  
    glClearColor (0.5, 0.5, 1, 1.0);  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity();  
    gluOrtho2D(-10,10,-5,5);  
    glClear (GL_COLOR_BUFFER_BIT);  
    // прорисовка изображения треугольника  
    glBegin(GL_TRIANGLES);  
    glColor3f(1,0,0); // задаем цвет вершин, чтобы треугольник  
    не слился с фоном  
    glVertex2d(-2.3); // первая вершина  
    glVertex2d(4.4); // вторая вершина  
    glVertex2d(7.-4); // третья вершина  
    glEnd();  
    // конец рисования треугольника  
    wglMakeCurrent(0, 0);  
end;
```

**Примечание.** Необходимо следить, чтобы изображение не выходило за пределы установленной координатной сетки (команды `gluOrtho2d` – для двумерных построений или `glOrtho` – для трехмерных построений).

## ПРАКТИЧЕСКАЯ РАБОТА №3. ИЗМЕНЕНИЕ ПРЕДОПРЕДЕЛЕННЫХ СВОЙСТВ ПРИМИТИВОВ

Приведенные в данном разделе команды, включающие свойства, необходимо написать **до** операторных скобок `glBegin()`. Команды выключения `glDisable()` необходимо указывать **после**

операторных скобок.

### **Точки.**

*Размер точки* устанавливается с помощью функции:

```
glPointSize(GLfloat size);
```

*Режим сглаживания точек* (по умолчанию точки квадратные) устанавливается вызовом функции

```
glEnable(GL_POINT_SMOOTH);
```

Отключается, соответственно, вызовом `glDisable()` с этим параметром. Последние функции – `glPointSize` и `glEnable/glDisable` надо вызывать вне `glBegin/glEnd`, иначе они будут проигнорированы.

### **Линии.**

*Ширина (толщина) линии:*

```
glLineWidth(int width)
```

*Прерывистые линии* получаются путем наложения маски при помощи следующей функции:

```
glLineStipple(GLint factor, GLushort pattern );
```

Второй параметр задает саму маску. Например, если его значение равно `255(0x00FF)`, то, чтобы вычислить задаваемую маску, воспользуемся калькулятором. В двоичном виде это число выглядит так: `0000000011111111`, т.е. всего 16 бит. Старшие восемь установлены в ноль, значит, тут линии не будет. Младшие установлены в единицу, тут будет рисоваться линия. Первый параметр определяет, сколько раз повторяется каждый бит. Скажем, если его установить равным 2, то накладываемая маска будет выглядеть так:

```
00000000000000001111111111111111
```

### **Пример 1.**

```
glLineWidth(1); // ширину линии устанавливаем 1
glBegin(GL_LINES);
    glColor3d(1,0,0); // красный цвет
    glVertex3d(-4.5,3,0); // первая линия
    glVertex3d(-3,3,0);
    glColor3d(0,1,0); // зеленый
    glVertex3d(-3,3.3,0); // вторая линия
    glVertex3d(-4,3.4,0);
glEnd();
glLineWidth(3); // ширина 3
glBegin(GL_LINE_STRIP);
```

## Моделирование объектов в OpenGL

```

glColor3d(1,0,0);
glVertex3d(-2.7,3,0);
glVertex3d(-1,3,0);
glColor3d(0,1,0);
glVertex3d(-1.5,3.3,0);
glColor3d(0,0,1);
glVertex3d(-1,3.5,0);
glEnd();
glLineWidth(5);
glEnable(GL_LINE_SMOOTH); // включаем сглаживание ли-
нии
glEnable(GL_LINE_STIPPLE); // разрешаем рисовать
прерывистую линию
glLineStipple(2,58360); // устанавливаем маску
glBegin(GL_LINE_LOOP);
glColor3d(1,0,0);
glVertex3d(1,3,0);
glVertex3d(4,3,0);
glColor3d(0,1,0);
glVertex3d(3,2.7,0);
glColor3d(0,0,1);
glVertex3d(2.5,3.7,0);
glEnd();
glDisable(GL_LINE_SMOOTH); //отключаем сглаживание
линии
glDisable(GL_LINE_STIPPLE); // запрещаем рисование
прерывистой линии
    
```

### ***Полигоны.***

Чтобы изменить *метод отображения* многоугольника используется команда:

*glPolygonMode (GLenum face, GLenum mode)*

Параметр *mode* определяет, как будут отображаться многоугольники, а параметр *face* устанавливает тип многоугольников, к которым будет применяться эта команда и могут принимать следующие значения:

Таблица 2. Значения параметров *face* и *mode*

GLenum face	<b>GL_FRONT</b>	для лицевых граней
	<b>GL_BACK</b>	для обратных граней
	<b>GL_FRONT_AND_BACK</b>	для всех граней



## Моделирование объектов в OpenGL

Glenum mode	<b>GL_POINT</b>	Отображаются вершины многоугольников
	<b>GL_LINE</b>	представляется набором отрезков
	<b>GL_FILL</b>	закрашиваются текущим цветом с учетом освещения. Этот режим установлен по умолчанию.

*Примечание.* Полигоны, нарисованные командой `GL_LINE_LOOP` нельзя закрасить.

### Пример 2.

```

glLineWidth(2);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE); //рисуем
проволочные треугольники
glBegin(GL_TRIANGLE_STRIP); // обратите внимание на
порядок вершин
    glColor3d(0,1,0);
    glVertex3d(1,2,0);
    glVertex3d(0,2.9,0);
    glVertex3d(-1,2,0);
    glVertex3d(0,1.1,0);
glEnd();
glEnable(GL_LINE_STIPPLE);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glBegin(GL_TRIANGLE_FAN);
    glColor3d(0,0,1);
    glVertex3d(4,2,0);
    glVertex3d(2.6,2.8,0);
    glVertex3d(2,2,0);
    glVertex3d(3,1.1,0);
glEnd();
glDisable(GL_LINE_STIPPLE);
    
```

*Модель заливки* фигур устанавливается оператором:

*glShadeModel (Glenum mode);*

Параметр *mode* может принимать значения `GL_SMOOTH` (плавная заливка – режим по умолчанию) или `GL_FLAT` (плоская заливка). При использовании плоской заливки цвет одной отдельной вершины независимого примитива дублируется для всех остальных вершин при визуализации этого примитива.

При плавной заливке цвет каждой вершины считается ин-

дивидуальным. Для линии цвет на протяжении отрезка интерполируется на основании цветов на его концах. Для полигона цвета его внутренней области интерполируются между цветами его вершин.

При использовании плавной заливки соседние пиксели имеют немного различающиеся цвета. В RGBA режиме соседние пиксели с немного различающимся цветом выглядят одинаково, таким образом, цвет плавно меняется по всей плоскости полигона.

## ПРАКТИЧЕСКАЯ РАБОТА №4. ПРОЦЕДУРА ДЛЯ РИСОВАНИЯ ОКРУЖНОСТИ И ПРАВИЛЬНЫХ МНОГОУГОЛЬНИКОВ

Окружность не является примитивом библиотеки OpenGL. Поэтому ее рассматривают как правильный многоугольник с достаточно малой длиной стороны, чтобы при отображении на экране он сглаживался и выглядел, как окружность.

Таким образом, задавая процедуру прорисовки окружности с помощью ее параметрических уравнений, эту же процедуру можно использовать для правильных многоугольников, вписанных в окружность данного радиуса.

В предложенной ниже процедуре используется динамический массив для возможности использования одной и той же процедуры для рисования в одной программе и окружностей, и различных многоугольников, а также их частей.

Аргументы процедуры (являются глобальными переменными):

***N*** – количество вершин многоугольника (или точность разбиения окружности),

***R*** – радиус окружности/описанной около многоугольника окружности,

***x<sub>0</sub>*** – абсцисса центра окружности/многоугольника,

***y<sub>0</sub>*** – ордината центра окружности/многоугольника,

***alpha<sub>0</sub>*** – угол поворота первой точки относительно положительного направления локальной оси *Ox* («+» - против часовой стрелки, «-» - по часовой стрелки). Если этот параметр равен 0, то первая точка расположена на локальной оси *Ox*.

***Beta*** – центральный угол дуги окружности. При  $\beta = 360^\circ$

## Моделирование объектов в OpenGL

имеем полную окружность.

Необходимо подключить модуль Math.

// процедура определения координат окружности/вершин  
 правильного многоугольника

**procedure** Coordinaty(n:integer;r,x0,y0, alpha0,beta: double);

**var**

alpha: double;

i:integer;

**begin**

// установка длины динамических массивов x и y (глобаль-  
 ные переменные)

SetLength(x, n);

SetLength(y, n);

for i := 0 to n-1 do

begin

alpha:=DegToRad(alpha0)+ DegToRad(beta)\*i/n; // шаг угла

между вершинами

// параметрические уравнения окружности

x[i]:=x0+r\*cos(alpha);

y[i]:=y0+r\*sin(alpha);

end;

**end;**

**procedure** TForm3.FormPaint(Sender: TObject); // процедура  
 отрисовки изображения

var i:integer;

begin

wglMakeCurrent(Canvas.Handle, hrc);

glClearColor(0.5,0.7,1,1);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-15,15,-15,15);

glClear(GL\_Color\_Buffer\_BIT);

// ВОСЬМИУГОЛЬНИК

n:=8;

Coordinaty (n,3,7,-7,0,360);

glBegin(gl\_polygon); // ПОЛИГОН

for i := 0 to n-1 do

begin

glColor3f(1,0,1);

## Моделирование объектов в OpenGL

```
    glVertex2d(x[i],y[i]);
end;
glEnd();

// окружность
n:=40;
Coordinaty (n,4,-6,-6,0,360);
glBegin(gl_line_loop); // замкнутая ломаная
for i := 0 to n-1 do
begin
    glColor3f(0.2,0.4,0);
    glVertex2d(x[i],y[i]);
end;
glEnd();

// шестиугольник, у которого две вершины лежат на оси Oy
n:=6;
Coordinaty (n,5,0,0,30,360);
glLineWidth(5); // толщина линии на экране
glBegin(gl_line_loop); // замкнутая ломаная
glColor3f(0.5,1,0.2); // цвет линии
for i := 0 to n-1 do
    glVertex2d(x[i],y[i]);
glEnd();

// дуга окружности с центральным углом 35° и первой точкой, повернутой на -45°.
n:=20;
Coordinaty (n,3,-2,-3,-45,35);
glBegin(gl_line_strip); //разомкнутая ломаная
for i := 0 to n-1 do
begin
    glColor3f(0,0.4,1);
    glVertex2d(x[i],y[i]);
end;
glEnd();

wglMakeCurrent(0,0);
end;
```

## ПРАКТИЧЕСКАЯ РАБОТА №5. ГРАФИКИ ФУНКЦИЙ

Для вывода на экран графика какой-либо функции необходимо прописать ее определение в отдельной процедуре, а затем ее вызвать.

**Пример.** Отобразить синусоиду на отрезке  $[0, 2\pi]$ .

```

procedure Grafic(n:integer);
var
alpha: double;
i:integer;
begin
SetLength(x, n);
SetLength(y, n);
  for i := 0 to n-1 do    // определение точек на графике
begin
alpha:=2*pi*i/n;
// уравнение синусоиды в параметрическом виде
x[i]:= alpha;
y[i]:= sin(x[i]);
end;
end;
    
```

**Задание.** Написать процедуры для вывода графика функций. Границы изменения параметра определить самостоятельно.

1	Гипербола $x = acht,$ $y = bsht$	6	Астроида $x = a \sin^3 \alpha,$ $y = a \cos^3 \alpha$
2	Строфоида $x = \frac{m(w^2 - 1)}{1 + w^2},$ $y = \frac{mw(w^2 - 1)}{1 + w^2}$	7	Дельтоида $x = 2m \cos \frac{\alpha}{3} + m \cos \frac{2\alpha}{3},$ $y = 2m \sin \frac{\alpha}{3} - m \sin \frac{2\alpha}{3}$
3	Циссоида $x = \frac{at^2}{1 + t^2},$ $y = \frac{at^3}{1 + t^2}$	8	Роза $x = m \sin(n\alpha) \cos \alpha,$ $y = m \sin(n\alpha) \sin \alpha$

4	Конхоида Никомеда $x = m + n \cos \alpha,$ $y = mtg\alpha + n \sin \alpha$	9	Полукубическая парабола $x = t^2,$ $y = at^3$
5	Улитка Паскаля $x = m \cos^2 \alpha + n \cos \alpha,$ $y = m \cos \alpha \sin \alpha + n \sin \alpha$	10	Лемниската Жероно $x = \cos \alpha,$ $y = \sin \alpha \cos \alpha$

## ПРАКТИЧЕСКАЯ РАБОТА №6. АФФИННЫЕ ПРЕОБРАЗОВАНИЯ

В OPENGL все преобразования над объектами производятся с помощью операций с матрицами.

Для сохранения и восстановления текущей матрицы используются функции **glPushMatrix()** и **glPopMatrix()**. Они записывают и восстанавливают текущую матрицу из стека, причем для каждого типа матриц стек свой. Для модельно-видовых матриц его глубина равна как минимум 32, для остальных – как минимум 2.

Функции **glPushMatrix()** и **glPopMatrix()** восстанавливают переменные состояния и используются, например, при аффинных преобразованиях. К аффинным преобразованиям относятся команды сдвиг, вращение, масштабирование. Без использования матриц можно получить неожиданный результат.

Сама матрица может быть создана с помощью следующих команд.

### Масштабирование

Преобразование масштабирования увеличивает или уменьшает размеры объекта.

Команда масштабирования

**glScale[f d]** (GLtype  $x$ , GLtype  $y$ , GLtype  $z$ )

с тремя аргументами – коэффициентами масштабирования по каждой из осей.

Если масштабные множители больше единицы объект растягивается в заданном направлении, если меньше объект сжимается. Масштабные множители могут иметь отрицательные значения, при этом изображение переворачивается по соответствующим

шей оси. При двумерных построениях значение коэффициента по оси Z игнорируется.

После команд рисования следует восстановить нормальный масштаб, чтобы каждое следующее обращение к обработчику перерисовки экрана не приводило бы к последовательному уменьшению / увеличению изображения.

### Поворот

Для поворота изображения используется команда

**glRotate[f d]** (GLtype *angle*, GLtype *x*, GLtype *y*, GLtype *z*)

с четырьмя аргументами:

GLtype *angle* – угол поворота (в градусах),

GLtype *x*, GLtype *y*, GLtype *z* – вектор поворота.

### Сдвиг (смещение)

Преобразование сдвига смещает точки в новые позиции в соответствии с заданным вектором смещения. Перенос системы координат осуществляется командой

**glTranslate[f d]** (GLtype *x*, GLtype *y*, GLtype *z*)

GLtype *x*, GLtype *y*, GLtype *z* – величины переноса по каждой из осей.

Для поворота вокруг произвольной фиксированной точки сначала нужно выполнить преобразование сдвига, совмещающую заданную фиксированную точку с началом координат, потом выполнить преобразование поворота вокруг начала координат, а затем обратное преобразование сдвига. Порядок манипуляции с системой координат: вначале перенос, затем поворот, по окончании рисования – в обратном порядке: поворот, затем перенос.

Все эти преобразования изменяют текущую матрицу, а поэтому применяются к примитивам, которые определяются **позже**. В случае, если надо, например, повернуть один объект сцены, а другой оставить неподвижным, удобно сначала сохранить текущую видовую матрицу в стеке командой `glPushMatrix()`, затем вызвать `glRotate()` с нужными параметрами, описать примитивы, из которых состоит этот объект, а затем восстановить текущую матрицу командой `glPopMatrix()`.

### Пример.

```
glClear(GL_COLOR_BUFFER_BIT);  
// начальное положение фигуры  
glBegin(GL_POLYGON);  
glColor3f(1,0,0);
```

## Моделирование объектов в OpenGL

```
glVertex2f(-25.0, -25.0);
glVertex2f(25.0, -25.0);
glVertex2f(25.0, 25.0);
glVertex2f(-25.0, 25.0);
glEnd( );
// фигура после трансформации
glPushMatrix();
glRotatef(25, 0.0, 0.0, 1.0); // поворот вокруг
точки (0,0) против часовой стрелки на угол 25°)
glColor3f(1.0, 1.0, 1.0);
glBegin(GL_POLYGON);
glColor3f(1,1,0);
glVertex2f(-25.0, -25.0);
glVertex2f(25.0, -25.0);
glVertex2f(25.0, 25.0);
glVertex2f(-25.0, 25.0);
glEnd( );
glPopMatrix();
```

## ПРАКТИЧЕСКАЯ РАБОТА №7. ОТОБРАЖЕНИЕ ТРЕХМЕРНЫХ ФИГУР

В основной библиотеке OpenGL не существует трехмерных примитивов. Все объемные объекты создаются на основе базовых примитивов, расположенных определенным образом в пространстве. Для использования таких примитивов, как сфера, параллелепипед, тор необходимо подключить библиотеку GLUT.

### Проективные преобразования в OpenGL

Для создания сцены необходимо задать область вывода объектов и задать способ проецирования.

Если область вывода не задана явно, то в OpenGL используется установленная по умолчанию зона в виде куба видимости  $2x2x2$  с началом координат в центре куба.

В составе OpenGL имеются две функции для задания перспективных проекций и одна для задания параллельных проекций. Каждая из функций определяет зону видимости – пирамиду или параллелепипед. Объекты, не попадающие в эту зону, отсекаются и не включаются в отображаемую сцену.



## Перспективные преобразования в OpenGL

Параметры пирамиды видимости задаются функцией **glFrustum()**, смысл аргументов которой поясняет рис. 1.

**glFrustum**(*GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far*);

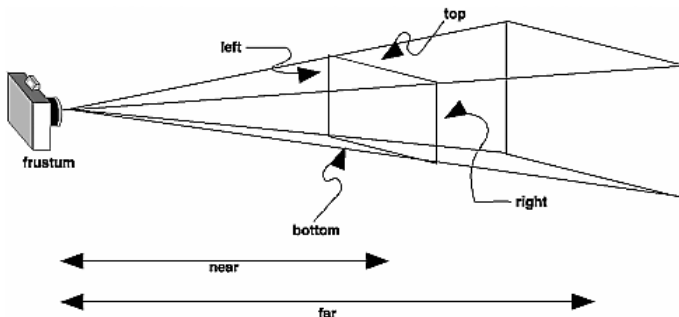


Рис. 1

Значения аргументов *near* и *far*, задающих положение передней и задней отсекающих плоскостей, должны быть положительными и отсчитываться от центра проецирования вдоль оси проецирования.

Поскольку матрица проецирования умножается на текущую матрицу, сначала нужно задать режим работы с этой матрицей. Типичная последовательность операций представлена ниже.

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(xmin, xmax, ymin, ymax, near, far);
```

Во многих приложениях предпочтительнее задавать не линейные параметры, характеризующие положение углов усеченной пирамиды видимости, а угол и поле зрения. Однако если картинная плоскость является прямоугольником, а не квадратом, то нужно задавать пару углов зрения: один в вертикальной плоскости, другой- в горизонтальной (рис. 2).

**gluPerspective**(*GLdouble fovy, GLdouble aspect, GLdouble znear, GLdouble zfar*);

Аргументы этой функции имеют следующий смысл:

*fovy* – угол зрения в вертикальной плоскости;

*aspect* – отношение ширины окна картинной плоскости к его высоте;

*znear* и *zfar* – расстояние от центра проецирования до пе-

редней и задней отсекающих плоскостей.

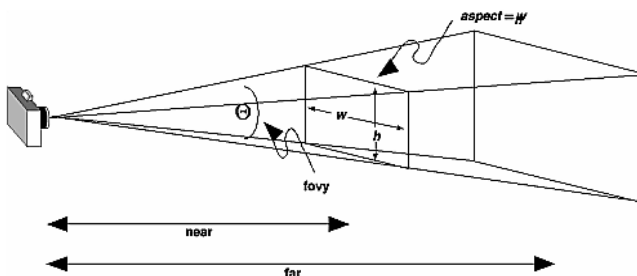


Рис. 2

Перспективная проекция задает усеченный конус видимости в левосторонней системе координат. Параметр **fovy** определяет угол видимости в градусах по оси **y** и должен находиться в диапазоне от 0 до 180. Угол видимости вдоль оси **x** задается параметром **aspect**, который обычно задается как отношение сторон области вывода (как правило, размеров окна). Параметры **zfar** и **znear** задают расстояние от наблюдателя до плоскостей отсечения по глубине и должны быть положительными. Чем больше отношение **zfar/znear**, тем хуже в буфере глубины будут различаться расположенные рядом поверхности, так как по умолчанию в него будет записываться 'сжатая' глубина в диапазоне от 0 до 1.

### Параллельное проецирование в OpenGL

В составе OpenGL имеется только одна функция для задания параметров параллельного проецирования, которая формирует ортогональную проекцию. Зона видимости при этом превращается в параллелепипед (рис. 3.).

**glOrtho**(*GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far*);

Аргументы вызова имеют тот же геометрический смысл, что и одноименные аргументы функции *glFrustum*().

## Моделирование объектов в OpenGL

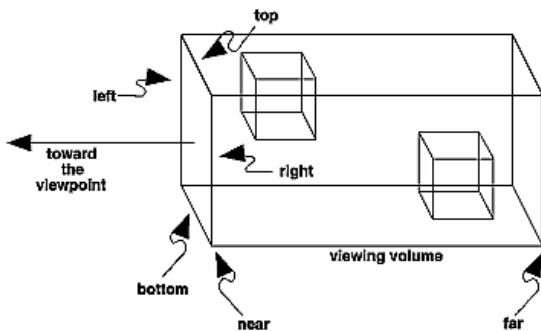


Рис. 3

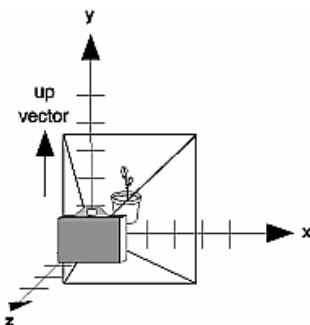
**Задание положения и ориентации камеры**


Рис. 4

В составе OpenGL имеется функция *gluLookAt()*, которая позволяет задать положение и ориентацию камеры (рис. 4).

***gluLookAt***(*GLdouble* *eyex*, *GLdouble* *eyez*, *GLdouble* *eyez*, *GLdouble* *centerx*, *GLdouble* *centery*, *GLdouble* *centerz*, *GLdouble* *upx*, *GLdouble* *upy*, *GLdouble* *upz*);

Аргументы функции имеют следующий вид:

*eyex*, *eyez*, *eyez* – координаты точки наблюдения;

*centerx*, *centery*, *centerz* –

координаты контрольной точки объекта, указывающей центр сцены;

*upx*, *upy*, *upz* - задает положительное направление оси *y*, определяя поворот камеры. Если, например, камеру не надо поворачивать, то задается значение (0,1,0), а со значением (0,-1,0) сцена будет перевернута.

Вызывать команду ***gluLookAt()*** имеет смысл *перед* определением преобразований объектов, когда модельно-видовая матрица равна единичной.

При изменении положения источника света следует учитывать следующий факт: в OpenGL источники света являются объектами, такими же, как многоугольники и точки. На них распространяется основное правило обработки координат в OpenGL –

параметры, описывающее положение в пространстве, преобразуются текущей модельно-видовой матрицей в момент формирования объекта, т.е. в момент вызова соответствующих команд OpenGL. Таким образом, формируя источник света одновременно с объектом сцены или камерой, его можно привязать к этому объекту. Или, наоборот, сформировать стационарный источник света, который будет оставаться на месте, пока другие объекты перемещаются.

#### **Общее правило такое:**

Если положение источника света задается командой `glLight*()` (будет рассмотрена ниже) перед определением положения виртуальной камеры (например, командой `glLookAt()`), то будет считаться, что координаты  $(0,0,0)$  источника находится в точке наблюдения и, следовательно, положение источника света определяется относительно положения наблюдателя.

Если положение устанавливается между определением положения камеры и преобразованиями модельно-видовой матрицы объекта, то оно фиксируется, т.е. в этом случае положение источника света задается в мировых координатах.

#### **Освещение**

В модели освещения OpenGL предполагается, что освещение может быть разделено на 4 компонента:

**фоновое (ambient)**

**диффузное (diffuse)**

**зеркальное (specular)**

**исходящее (эмиссионное – emissive).**

Все 4 компонента рассчитываются независимо и только затем суммируются.

**Чтобы добавить на сцену освещение, требуется выполнить несколько шагов:**

1. Определить вектор нормали для каждой вершины каждого объекта. Эти нормали задают ориентацию объекта по отношению к источникам света.

2. Создать, выбрать и позиционировать один или более источников света.

3. Создать и выбрать **модель освещения**, которая определяет уровень глобального фонового света и эффективное положение точки наблюдения (для вычислений, связанных с освещением).

4. Задать свойства материала для объектов сцены.

## RGB – величины для света и материалов

Цветовые компоненты, задаваемые для источников света, означают совсем не то же самое, что для материалов. Для источника света число представляет собой процент от полной интенсивности каждого цвета. Если R, G и B – величины цвета источника света все равны 1.0, свет будет максимально белым. Если величины будут равны 0.5, свет все равно останется белым, но лишь с половиной интенсивности (он будет казаться серым). Если R=G=1 и B=0 (полный красный, полный зеленый, отсутствие синего), свет будет желтым.

Для материалов числа соответствуют отраженным пропорциям этих цветов. Так что, если для материала R=1, G=0.5 и B=0, этот материал отражает весь красный свет, половину зеленого и совсем не отражает синего.

Если два источника света с характеристиками (R1, G1, B1) и (R2, G2, B2) направлены в глаз, OpenGL сложит компоненты: (R1+R2, G1+G2, B1+B2). Если какая-либо из сумм будет больше 1 (соответствуя цвету, который нельзя отобразить), компонент будет урезан до 1.

**Вызовы команд, связанных с освещением, помещают в функцию Create.**

```
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
/* устанавливаем параметры источника света */
glLightfv (GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv (GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv (GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv (GL_LIGHT0, GL_POSITION, light_position);
/* включаем освещение и источник света */
glEnable (GL_LIGHTING);
glEnable (GL_LIGHT0);
/* включаем z-буфер */
glEnable(GL_DEPTH_TEST);
```

**Создание, позиционирование и включение одного или более источников света**

### Создание источников света

Источники света имеют несколько параметров, таких как цвет, позиция и направление. Команда, используемая для

## Моделирование объектов в OpenGL

указания всех параметров света – это `glLight*()`. Она принимает три аргумента: идентификатор источника света, имя свойства и желаемое для него значение.

`glLight{if}` (GLenum *light*, GLenum *pname*, TYPE *param*);  
`glLight{if}v` (GLenum *light*, GLenum *pname*, TYPE \**param*);

Создает источник, задаваемый параметром *light* (который может принимать значения `GL_LIGHT0`, `GL_LIGHT1`, ..., `GL_LIGHT7`). Задаваемая характеристика света определяется аргументом *pname* в виде константы (таблица 3). В параметре *param* задается значение или значения, в которые следует установить характеристику *pname*.

Если используется векторная версия команды, *param* представляет собой вектор величин, а если не векторная, то *param* – одно единственное значение. Невекторная версия команды может использоваться только для указания параметров, чье значение выражается одним числом. По умолчанию цвет всех источников света кроме `GL_LIGHT0` – черный.

После настройки параметров источника света необходимо активизировать его командой `glEnable()`. Кроме того, необходимо вызвать команду `glEnable()` с аргументом `GL_LIGHTING`, чтобы подготовить OpenGL к выполнению расчетов, связанных с освещением.

*Замечание:* Помните, что каждый источник света нужно включить командой `glEnable()`.

Таблица 3. Значения по умолчанию для источника света

Имена параметров	Значения по умолчанию	Смысл
<code>GL_AMBIENT</code>	(0.0,0.0,0.0,1.0)	Интенсивность фонового света
<code>GL_DIFFUSE</code>	(1.0,1.0,1.0,1.0) или (0.0,0.0,0.0,1.0)	Интенсивность диффузного света (значение по умолчанию для 0-го источника - белый свет, для остальных - черный)
<code>GL_SPECULAR</code>	(1.0,1.0,1.0,1.0) или (0.0,0.0,0.0,1.0)	Интенсивность зеркального света (значение по умолчанию для 0-го источника - белый свет, для остальных - черный)
<code>GL_POSITION</code>	(0.0,0.0,1.0,0.0)	Положение источника света ( <i>x,y,z,w</i> )
<code>GL_SPOT_DIRECTION</code>	(0.0,0.0,-1.0)	Направление света прожектора ( <i>x,y,z</i> )
<code>GL_SPOT_EXPONENT</code>	0.0	Концентрация светового луча
<code>GL_SPOT_CUTOFF</code>	180.0	Угловая ширина светового луча
<code>GL_CONSTANT_ATTENUATION</code>	1.0	Постоянный фактор ослабления
<code>GL_LINEAR_ATTENUATION</code>	0.0	Линейный фактор ослабления
<code>GL_QUADRATIC_ATTENUATION</code>	0.0	Квадратичный фактор ослабления

**Замечание:**

Значения по умолчанию для `GL_DIFFUSE` и `GL_SPECULAR` в таблице различаются для `GL_LIGHT0` и других источников света (`GL_LIGHT1`, `GL_LIGHT2`,...). Для параметров `GL_DIFFUSE` и `GL_SPECULAR` источника света `GL_LIGHT0` значение по умолчанию – (1.0, 1.0, 1.0, 1.0). Для других источников света значение тех же параметров по умолчанию – (0.0, 0.0, 0.0, 1.0).

**Выбор модели освещения**

Параметры модели освещения описываются командой `glLightModel*()`. Модель освещения также позволяет указывать, где находится предполагаемое местоположение наблюдателя: бесконечно далеко или локально по отношению к сцене, и должны ли вычисления производиться по-разному для лицевых и обратных поверхностей объектов. Возможно использовать значения по умолчанию, например, для двух аспектов модели – наблюдатель находится бесконечно далеко (режим «бесконечно удаленного наблюдателя») и одностороннее освещение. Использование режима «локального наблюдателя» заставляет производить намного больший объем сложных расчетов, так как OpenGL должна вычислить угол между точкой наблюдения и каждым объектом. В режиме «бесконечно удаленного наблюдателя», однако, этот угол игнорируется, и результат может быть менее реалистичным. Поскольку в примере обратная поверхность сферы никогда не видна (она находится внутри сферы), достаточно одностороннего освещения.

`glLightModel*()` – это команда, используемая для задания всех параметров модели освещения, `glLightModel*()` принимает два аргумента: имя параметра модели освещения в виде константы и значение для этого параметра.

`glLightModel{if}` (GLenum *pname*, TYPE *param*);

`glLightModel{if}v` (GLenum *pname*, TYPE *\*param*);

Устанавливает свойства модели освещения. Устанавливаемая характеристика модели освещения определяется аргументом *pname* (таблица 4). *param* задает величину, в которую устанавливается *pname*; если используется векторная версия команды, то это указатель на группу величин, если применяется не векторная версия – в *param* содержится сама величина. Не векторная версия команды может использоваться только для установки параметров, определяемых одной величиной (и не может применяться для `GL_LIGHT_MODEL_AMBIENT`).

Таблица 4. Значения по умолчанию для параметра *pname* модели освещения

Имена параметров	Значения по умолчанию	Смысл
GL_LIGHT_MODEL_AMBIENT	(0.2,0.2,0.2,1.0)	RGBA интенсивность всей сцены
GL_LIGHT_MODEL_LOCAL_VIEWER	0.0 или GL_FALSE	способ вычисления углов зеркального отражения
GL_LIGHT_MODEL_TWO_SIDE	0.0 или GL_FALSE	выбор между односторонним и двухсторонним освещением
GL_LIGHT_MODEL_COLOR_CONTROL	GL_SINGLE_COLOR	вычисляется ли зеркальный цвет отдельно от фонового и диффузного

### Определение свойств материала для объектов сцены

Свойства материала объектов определяют, как он отражает свет и, таким образом, из какого реального материала он сделан (в зрительном восприятии). Поскольку взаимодействие между поверхностью материала и входящим светом достаточно сложное, довольно трудно задать такие параметры материала, чтобы объект имел определенный, желаемый вид. Можно задавать фоновый, диффузный и зеркальный цвета материала и то, насколько блестящим он будет выглядеть.

### Указание свойств материала

Большинство свойств материала похожи на те, которые использовались при создании источников света. Механизм из указания также аналогичен установке параметров источника света за исключением того, что здесь используется команда `glMaterial*()`.

```
glMaterial{if}(GLenum face, GLenum pname, TYPE param);
```

```
glMaterial{if}v(GLenum face, GLenum pname, TYPE *param);
```

Задаёт свойство материала для использования при расчете освещенности. Аргумент *face* может принимать значения `GL_FRONT`, `GL_BACK` или `GL_FRONT_AND_BACK`, указывая для каких граней объекта задается свойство материала. Устанавливаемое свойство материала определяется значением аргумента *pname*, а его значение содержится в *param* (в виде указателя на вектор величин в случае векторной версии команды или в виде самой величины при использовании не векторного варианта).

Невекторная версия команды работает только для параметра `GL_SHININESS`.

Возможные значения для аргумента *pname* перечислены в таблице 5. Заметьте, что константа `GL_AMBIENT_AND_DIFFUSE` позволяет одновременно установить фоновый и диффузный цвета



## Моделирование объектов в OpenGL

материала в одно и то же RGBAзначение.

Таблица 5. Значения по умолчанию для *pname*

Имена параметров	Значения по умолчанию	Смысл
GL_AMBIENT	(0.2,0.2,0.2,1.0)	фоновый цвет материала
GL_DIFFUSE	(0.8,0.8,0.8,1.0)	диффузный цвет материала
GL_AMBIENT_AND_DIFFUSE		фоновый и диффузный цвет материала
GL_SPECULAR	(0.0,0.0,0.0,1.0)	зеркальный цвет материала
GL_SHININESS	0.0	показатель зеркального отражения
GL_EMISSION	(0.0,0.0,0.0,1.0)	исходящий цвет материала
GL_COLOR_INDEXES	(0,1,1)	индексы фонового, диффузного и зеркального цветов

Можно задать расчет освещенности для лицевых и обратных полигонов объекта. Если в приложении обратные грани могут быть видимыми, можно по-разному задать параметры материала для лицевых и обратных граней объекта, используя аргумент *face* команды `glMaterial*()`.

**Пример.** Пример построения куба из четырехугольников средствами стандартной библиотеки.

*Замечание.* Используется второй вариант инициализации OpenGL. В примере даны только те процедуры, которые изменены.

```

procedure TForm6.FormCreate(Sender: TObject);
begin
    ghDC := GetDC(Handle);
    if not bSetupPixelFormat(ghDC) then Close();
    ghRC := wglCreateContext(ghDC);
    wglMakeCurrent(ghDC, ghRC);
    glClearColor (0.5, 0.5, 1, 1.0);
    FormResize(Sender);
    glEnable(GL_COLOR_MATERIAL); //включен материал
    glEnable(GL_LIGHTING); // включение источника света
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST); // включение буфера глубины
    glDepthMask(GL_TRUE);
    // параметры источника освещения
    pp[0] := -0.8;
    pp[1] := 1;
    pp[2] := 0.9;
    pp[3] := 0.5;

```

## Моделирование объектов в OpenGL

```
        dd[0] := 0.5;
        dd[1] := -1;
        dd[2] := 0.9;
        glLightfv(GL_LIGHT0, GL_POSITION, @pp);
        glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, @dd);
end;

procedure TForm6.Draw;
begin
    glLightfv(GL_LIGHT0, GL_POSITION, @pp); // при вызове
    этих 2-х функций, координаты pp и dd пересчитываются, умножа-
    юсь на видовую матрицу
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, @dd);
    glClear(GL_COLOR_BUFFER_BIT);
    glClear(GL_DEPTH_BUFFER_BIT);
    // куб
    glBegin (GL_QUADS) ;
    glColor3f(0,0,1);
    glVertex3f (1.0, 1.0, 1.0);
    glVertex3f (-1.0, 1.0, 1.0); //1 сторона
    glVertex3f (-1.0, -1.0, 1.0);
    glVertex3f (1.0, -1.0, 1.0);

    glVertex3f (1.0, 1.0, -1.0);
    glVertex3f (1.0, -1.0, -1.0);
    glVertex3f (-1.0, -1.0, -1.0); //2 сторона
    glVertex3f (-1.0, 1.0, -1.0);

    glVertex3f (-1.0, 1.0, 1.0);
    glVertex3f (-1.0, 1.0, -1.0) ;
    glVertex3f (-1.0, -1.0, -1.0) ; //3 сторона
    glVertex3f (-1.0, -1.0, 1.0);

    glVertex3f (1.0, 1.0, 1.0);
    glVertex3f (1.0, -1.0, 1.0); //4 сторона
    glVertex3f (1.0, -1.0, -1.0);
    glVertex3f (1.0, 1.0, -1.0);

    glVertex3f (-1.0, 1.0, -1.0);
    glVertex3f (-1.0, 1.0, 1.0); //5 сторона
    glVertex3f (1.0, 1.0, 1.0);
```

## Моделирование объектов в OpenGL

```
glVertex3f (1.0, 1.0, -1.0);  
  
glVertex3f (-1.0, -1.0, -1.0);  
glVertex3f (1.0, -1.0, -1.0); //6 сторона  
glVertex3f (1.0, -1.0, 1.0);  
glVertex3f (-1.0, -1.0, 1.0);  
glEnd();  
  
SwapBuffers(ghDC);  
  
end;
```

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Краснов М. В. OpenGL. Графика в проектах Delphi. — СПб.: БХВ-Петербург, 2002. - 352 с.
2. Ю.М. Баяковский, А.В. Игнатенко. Начальный курс OpenGL. М: «Планета знаний», 2007. – 221 с.
3. Р. С. Райт., Б. Липчак. OpenGL. Суперкнига. 3-е издание. М: Издательский дом «Вильямс», 2006. – 1040 с.
4. Д.Л. Осипов. Delphi. Программирование для Windows, OS X, iOS и Android. 2014. – 464 с.